

Hertrich, Christoph; Weiß, Christian; Ackermann, Heiner; Heydrich, Sandy;
Krumke, Sven O.

Article — Published Version

Scheduling a proportionate flow shop of batching machines

Journal of Scheduling

Provided in Cooperation with:

Springer Nature

Suggested Citation: Hertrich, Christoph; Weiß, Christian; Ackermann, Heiner; Heydrich, Sandy; Krumke, Sven O. (2020) : Scheduling a proportionate flow shop of batching machines, Journal of Scheduling, ISSN 1099-1425, Springer US, New York, NY, Vol. 23, Iss. 5, pp. 575-593, <https://doi.org/10.1007/s10951-020-00667-2>

This Version is available at:

<https://hdl.handle.net/10419/288444>

Standard-Nutzungsbedingungen:

Die Dokumente auf EconStor dürfen zu eigenen wissenschaftlichen Zwecken und zum Privatgebrauch gespeichert und kopiert werden.

Sie dürfen die Dokumente nicht für öffentliche oder kommerzielle Zwecke vervielfältigen, öffentlich ausstellen, öffentlich zugänglich machen, vertreiben oder anderweitig nutzen.

Sofern die Verfasser die Dokumente unter Open-Content-Lizenzen (insbesondere CC-Lizenzen) zur Verfügung gestellt haben sollten, gelten abweichend von diesen Nutzungsbedingungen die in der dort genannten Lizenz gewährten Nutzungsrechte.

Terms of use:

Documents in EconStor may be saved and copied for your personal and scholarly purposes.

You are not to copy documents for public or commercial purposes, to exhibit the documents publicly, to make them publicly available on the internet, or to distribute or otherwise use the documents in public.

If the documents have been made available under an Open Content Licence (especially Creative Commons Licences), you may exercise further usage rights as specified in the indicated licence.



<https://creativecommons.org/licenses/by/4.0/>



Scheduling a proportionate flow shop of batching machines

Christoph Hertrich^{1,2} · Christian Weiß³ · Heiner Ackermann³ · Sandy Heydrich³ · Sven O. Krumke¹

Published online: 17 September 2020
© The Author(s) 2020

Abstract

In this paper we study a proportionate flow shop of batching machines with release dates and a fixed number $m \geq 2$ of machines. The scheduling problem has so far barely received any attention in the literature, but recently its importance has increased significantly, due to applications in the industrial scaling of modern bio-medicine production processes. We show that for any fixed number of machines, the makespan and the sum of completion times can be minimized in polynomial time. Furthermore, we show that the obtained algorithm can also be used to minimize the weighted total completion time, maximum lateness, total tardiness and (weighted) number of late jobs in polynomial time if all release dates are 0. Previously, polynomial time algorithms have only been known for two machines.

Keywords Planning of pharmaceutical production · Proportionate flow shop · Batching machines · Permutation schedules · Dynamic programming

1 Introduction

Modern medicine can treat some serious illnesses using individualized drugs, which are produced to order for a specific patient and adapted to work only for that unique patient and nobody else. Manufacturing such a drug often involves a complex production line, consisting of many different steps.

If each step of the process is performed manually, by a laboratory worker, then each laboratory worker can only handle materials for one patient at a time. However, in the industrial

application which motivates our study, some steps are instead performed by actual machines, like pipetting robots. Such high-end machines can typically handle materials for multiple patients in one go. If scheduled efficiently, this special feature can drastically increase the throughput of the production line. Clearly, in such an environment efficient operative planning is crucial in order to optimize the performance of the manufacturing process and treat as many patients as possible.

In this paper, we present – as part of a larger study of an industrial application – new theoretical results for scheduling problems arising from a manufacturing process as described above. Formally, the manufacturing process we consider is structured in a *flow shop* manner, where each step is handled by a single, dedicated machine. A job J_j , $j = 1, 2, \dots, n$, representing the production of a drug for a specific patient, has to be processed by machines M_1, M_2, \dots, M_m in order of their numbering. Each job J_j has a *release date* $r_j \geq 0$, denoting the time at which the job J_j is available for processing at the first machine M_1 . Furthermore, a job is only available for processing at machine M_i , $i = 2, 3, \dots, m$, when it has finished processing on the previous machine M_{i-1} .

Processing times are job-independent, meaning that each machine M_i , $i = 1, 2, \dots, m$, has a fixed *processing time* p_i , which is the same for every job when processed on that machine. In the literature, a flow shop with machine-

✉ Christian Weiß
christian.weiss@itwm.fraunhofer.de

Christoph Hertrich
hertrich@math.tu-berlin.de

Heiner Ackermann
heiner.ackermann@itwm.fraunhofer.de

Sandy Heydrich
sandy.heydrich@itwm.fraunhofer.de

Sven O. Krumke
krumke@mathematik.uni-kl.de

¹ Department of Mathematics, Technische Universität Kaiserslautern, 67663 Kaiserslautern, Germany

² Institute of Mathematics, Technische Universität Berlin, 10623 Berlin, Germany

³ Department of Optimization, Fraunhofer Institute for Industrial Mathematics ITWM, 67663 Kaiserslautern, Germany

or job-independent processing times is sometimes called a *proportionate* flow shop, see e.g. (Panwalkar et al. 2013).

Recall that, as a special feature from our application, each machine in the flow shop can handle multiple jobs at the same time. These kind of machines are called (*parallel*) *batching machines* and a set of jobs processed at the same time on some machine is called a *batch* on that machine (Brucker 2007, Chapter 8). All jobs in one batch $B_k^{(i)}$ on some machine M_i have to start processing on M_i at the same time. In particular, all jobs in $B_k^{(i)}$ have to be available for processing on M_i , before $B_k^{(i)}$ can be started. The processing time of a batch on M_i remains p_i , no matter how many jobs are included in this batch. This distinguishes parallel batching machines from *serial batching machines*, where the processing time of a batch is calculated as the sum of the processing times of the jobs contained in it plus an additional setup time. Each machine $M_i, i = 1, 2, \dots, m$ has a *maximum batch size* b_i , which is the maximum number of jobs a batch on machine M_i may contain.

Given a feasible schedule S , we denote by $c_{ij}(S)$ the *completion time* of job J_j on machine M_i . For the completion time of job J_j on the last machine we also write $C_j(S) = c_{mj}(S)$. If there is no confusion which schedule is considered, we may omit the reference to the schedule and simply write c_{ij} and C_j .

As optimization criteria, in this paper we are interested in objective functions of the forms

$$f_{\max} = \max_{j=1,2,\dots,n} f_j(C_j), \tag{1}$$

$$f_{\Sigma} = \sum_{j=1}^n f_j(C_j), \tag{2}$$

with functions f_j non-decreasing for $j = 1, 2, \dots, n$.

In particular, the first part of the paper will be dedicated to the minimization of the *makespan* $C_{\max} = \max \{C_j \mid j = 1, 2, \dots, n\}$ and of the *total completion time* $\sum C_j = \sum_{j=1}^n C_j$, although the obtained algorithms will work for an arbitrary objective function of type (1) or (2), if certain pre-conditions are met.

In the second part of the paper, we assume that for each job J_j a *weight* w_j or a *due date* d_j is given. We focus on the following traditional scheduling objectives:

- the *weighted total completion time*
 $\sum w_j C_j = \sum_{j=1}^n w_j C_j,$
- the *maximum lateness*
 $L_{\max} = \max \{C_j - d_j \mid j = 1, 2, \dots, n\},$
- the *total tardiness*
 $\sum T_j = \sum_{j=1}^n T_j,$
- the *number of late jobs*
 $\sum U_j = \sum_{j=1}^n U_j,$

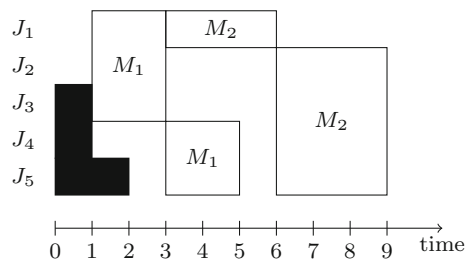


Fig. 1 A feasible example schedule

- the *weighted number of late jobs*
 $\sum w_j U_j = \sum_{j=1}^n w_j U_j,$

where

$$T_j = \begin{cases} C_j - d_j, & \text{if } C_j > d_j, \\ 0, & \text{otherwise,} \end{cases}$$

and

$$U_j = \begin{cases} 1, & \text{if } C_j > d_j, \\ 0, & \text{otherwise.} \end{cases}$$

Note that all these objective functions are *regular*, that is, nondecreasing in each job completion time C_j .

Using the standard three-field notation for scheduling problems (Graham et al. 1979), our problem is denoted as

$$Fm \mid r_j, p_{ij} = p_i, p\text{-batch}, b_i \mid f,$$

where f is a function of the form defined above. We refer to the described scheduling model as *proportionate flow shop of batching machines* and abbreviate it by *PFB*.

Next, we provide an example in order to illustrate the problem setting.

Example 1 Consider a PFB instance with $m = 2$ machines, $n = 5$ jobs, maximum batch sizes $b_1 = 3$ and $b_2 = 4$, processing times $p_1 = 2$ and $p_2 = 3$, and release dates $r_1 = r_2 = 0, r_3 = r_4 = 1$, and $r_5 = 2$. Figure 1 illustrates a feasible schedule for the instance as job-oriented Gantt chart.

Each rectangle labeled by a machine represents a batch of jobs processed together on this machine. The black area indicates that the respective jobs have not been released at this time yet. Note that in this example none of the batches can be started earlier, since either a job of the batch has just arrived when the batch is started, or the machine is occupied before. Still, the schedule does not minimize the makespan, since the schedule shown in Fig. 2 is feasible as well and has a makespan of 8 instead of 9.

The improvement in the makespan was achieved by reducing the size of the first batch on M_1 from three to two, which allows to start it one time step earlier. Observe that no job

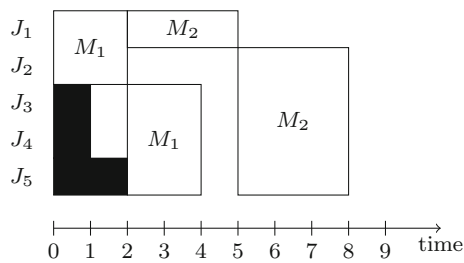


Fig. 2 An example schedule minimizing the makespan

can finish M_2 before time step 5. Moreover, not all five jobs fit into one batch on M_2 . Hence, at least two batches are necessary on M_2 in any schedule. Therefore, 8 is the optimal makespan and no further improvement is possible.

The rest of the paper is structured as follows. The remaining parts of this section provide an overview of the related literature and our results, respectively. In Sect. 2 we prove that permutation schedules with jobs in earliest release date order are optimal for PFBs with the makespan and total completion time objectives. We also show that permutation schedules are not optimal for any other traditional scheduling objective functions. In Sect. 3 we construct a dynamic program to find the optimal permutation schedule with a given, fixed job order. We show that, if the number of machines m is fixed, the dynamic program can be used to minimize the makespan or total completion time in a PFB in polynomial time. In Sect. 4 we consider PFBs where all release dates are equal. We show that, in this special case, permutation schedules are always optimal and that the dynamic program from Sect. 3 can be applied to solve most traditional scheduling objectives. Finally, in Sect. 5 we draw conclusions, reflect on our results in the context of some additional existing literature, and point out future research directions.

1.1 Literature

In this overview as well as the remainder of this paper, we are exclusively concerned with flow shops of parallel batching machines, which we will simply call batching machines. For complexity results about (proportionate) flow shops with serial batching machines, we refer e.g. to Ng and Kovalyov (2007) and Brucker and Shakhlevich (2011). It is interesting to note that, although the different type of batching machines changes the problem significantly, the methodology used by Ng and Kovalyov (2007) is quite similar to ours: for a fixed job permutation a dynamic program is developed that adds jobs one by one, resulting in polynomial runtime for a fixed number of machines.

The proportionate flow shop problem with batching machines as studied in this paper has so far not received a lot of attention from researchers, although several appli-

cations have appeared in the literature. In addition to the application from pharmacy described in Sect. 1, Sung et al. (2000) mention a variety of applications in the manufacturing industry, e.g., in the production of semi-conductors. Furthermore, there are several papers which study the scheduling sequences of locks for ships along a canal or river, which can in some ways be seen as a generalization of PFBs (see, e.g., Passchyn and Spieksma (2019)). The problem is generally strongly NP-complete, but several special cases for which it can be solved in polynomial time have been identified.

Another stream of recent research focuses on finding practical scheduling techniques for production processes which can be viewed as NP-hard generalizations of the PFB problem. Usually application-specific heuristics are developed and their quality is certified by computational studies. A preliminary study of the specific industrial setup which motivated our research was done in Ackermann et al. (2020+). For studies from different areas of application we refer, for instance, to Li et al. (2015), Zhou et al. (2016), Tan et al. (2018), Chen et al. (2019), Li et al. (2019). The problems typically stem from various industries, e.g. electronics, steel, or equipment manufacturing. Features that make these problems difficult include mixtures of parallel and serial batching machines, families of jobs that are incompatible to be batched together, hybrid/flexible flow shop setups, as well as no-wait constraints.

Despite this multitude of possible applications, for PFBs as introduced in Sect. 1 significant hardness results have, to the best of our knowledge, not been achieved at all. Polynomial time algorithms are only known for the special case with no more than two machines and, in most cases, without release dates (Ahmadi et al. 1992; Sung and Yoon 1997; Sung and Kim 2003).

The paper by Sung et al. (2000) is, as far as we know, the only work in literature considering a PFB with arbitrarily many machines. They propose a reduction procedure to decrease the size of an instance by eliminating dominated machines. Using this procedure, they develop heuristics to minimize the makespan and the sum of completion times in a PFB and conduct a computational study certifying quality and efficiency of their heuristic approach. However, they do not establish any complexity result.

For the case with $m = 2$ machines and no release dates, Ahmadi et al. (1992) show that the makespan can be minimized by completely filling all batches on the first machine (other than the last one, if the number of jobs is not a multiple of b_1) and completely filling all batches on the second machine (other than the first one, if the number of jobs is not a multiple of b_2). This structural result immediately yields an $\mathcal{O}(n)$ time algorithm. If release dates do not all vanish, i.e. $r_j > 0$ for some $j = 1, 2, \dots, n$, then the makespan can be minimized in $\mathcal{O}(n^2)$ time, see Sung and Yoon (1997). The authors use a dynamic programming algorithm in order to

find an optimal batching on the first machine. On the second machine, the strategy to completely fill all batches (other than maybe the first one) is still optimal.

For objectives other than the makespan, results are only known if all jobs are released at the same time. In this case, the total completion time in a two-machine PFB can be minimized in $\mathcal{O}(n^3)$ time, see (Ahmadi et al. 1992). Sung and Kim (2003) provide dynamic programs to minimize L_{\max} , $\sum T_j$, and $\sum U_j$ in a two-machine PFB without release dates in polynomial time.

There are two special cases and one generalization of PFB which have been studied relatively extensively in the literature and which are of importance when it comes to comparison with the wider literature. First, consider the special case of PFB with maximum batch size equal to one on all machines. This is the usual proportionate flow shop problem, which is well-researched and known to be polynomially solvable for many variants. An overview is given in Panwalkar et al. (2013). In particular, the makespan and the total completion time can be minimized in $\mathcal{O}(n \log n + nm)$ time, by ordering jobs according to nondecreasing release dates and scheduling each job as early as possible on each machine. Note that the term $n \log n$ is due to the sorting of the jobs while the term nm arises from computing start and completion times for each operation of each job.

Second, consider the special case of PFB where the number of machines is fixed to $m = 1$, which leaves us with the problem of scheduling a single batching machine with identical (but not necessarily unit) processing times. This problem, too, has been investigated by various authors (Ikura and Gimple 1986; Lee et al. 1992; Baptiste 2000; Condotta et al. 2010). Even with release dates, the problem can be solved in polynomial time for almost all standard scheduling objective functions, except for the weighted total tardiness, for which it is open. Note, in particular, the reduction procedure from Sung et al. (2000) implies that for the cases where all batch sizes or all processing times are equal, $b_i = \bar{b}$ or $p_i = \bar{p}$ for all $i = 1, 2, \dots, m$, the problem of scheduling a PFB reduces to scheduling a single batching machine with equal processing times. Thus, for those two special cases, the results by Baptiste (2000) imply that scheduling a PFB is polynomially solvable for all objective functions other than the weighted total tardiness.

Third, in contrast to the two special cases from before, consider the generalization of PFB where processing times may be job- as well as machine-dependent. We end up with the usual flow shop problem with batching machines. Observe that this problem is also a generalization of traditional flow shop as well as scheduling a single batching machine. Therefore, we can immediately deduce that, even without release dates, it is strongly NP-hard to minimize the makespan for any fixed number of machines $m \geq 3$ and to minimize the total completion time for any fixed number of machines

$m \geq 2$ (Garey et al. 1976). Furthermore, minimizing the maximum lateness (or, equivalently, the makespan with release dates) is strongly NP-hard even for the case with $m = 1$ and $b_1 = 2$ (Brucker et al. 1998). Finally, Potts et al. (2001) showed that scheduling a flow shop with batching machines to minimize the makespan, without release dates, is strongly NP-hard, even for $m = 2$ machines. Recall that for traditional flow shop with $m = 2$ machines a schedule with minimum makespan can be computed in $\mathcal{O}(n \log n)$ time (Johnson 1954).

In conclusion, we see that dropping either the batching or the multi-machine property of PFB leads to easy special cases, while dropping the proportionate property leads to a very hard generalization. Therefore, PFB lies exactly on the borderline between easy and hard problems, which makes it an interesting problem to study from a theoretical perspective, in addition to the practical considerations explained in the introduction.

1.2 Our results

Our main focus in this paper is the complexity of PFBs with an arbitrary, but fixed number m of machines. This is motivated by the specific industrial application we consider: while the number of steps needed to produce individualized drugs in a process as described in the beginning is significantly larger than 2 or 3 (machine numbers which are often studied separately in the literature), it is not arbitrarily large and it does not change once the process is set up. Thus, it is reasonable to assume that m is fixed and not part of the input.

This paper is, to the best of our knowledge, the first study which provides any complexity results – positive or negative – for PFBs with more than $m = 2$ machines (see Table 1 for an overview). As such we close several questions which have remained open for over fifteen years, despite the significant practical importance of PFBs in the area of production lines. We show that for PFBs with release dates and a fixed number m of machines, the makespan and the total completion time can be minimized in polynomial time. For PFBs without release dates, we further show that the weighted total completion time, the maximum lateness, the total tardiness and the (weighted) number of late jobs can be minimized in polynomial time.

Note that, while this paper closes many open complexity questions for PFBs, the proposed algorithm has practical limitations concerning its running time. We discuss this in some more detail also in the conclusions to this paper. Still, some of the structural results proved in this paper, in particular concerning the optimality of permutation schedules proved in Sects. 2 and 4, can be of great help for the future design of heuristics or approximation algorithms, which run fast in practice. Indeed, for the specific industrial setup of the study this paper originates from, such practical heuristics

Table 1 Polynomially solvable PFB variants

Problem	$m = 1$	$m = 2$	fixed $m \geq 3$
$Fm \mid r_j, p_{ij} = p_i, p\text{-batch}, b_i \mid C_{\max}$	Ikura and Gimple (1986)	Sung and Yoon (1997)	Corollary 13/14
$Fm \mid r_j, p_{ij} = p_i, p\text{-batch}, b_i \mid \sum C_j$	Ahmadi et al. (1992)	Corollary 13	Corollary 13
$Fm \mid p_{ij} = p_i, p\text{-batch}, b_i \mid \sum w_j C_j$	Baptiste (2000)	Theorem 18	Theorem 18
$Fm \mid p_{ij} = p_i, p\text{-batch}, b_i \mid L_{\max}$	Lee et al. (1992)	Sung and Kim (2003)	Theorem 18
$Fm \mid p_{ij} = p_i, p\text{-batch}, b_i \mid \sum T_j$	Baptiste (2000)	Sung and Kim (2003)	Theorem 18
$Fm \mid p_{ij} = p_i, p\text{-batch}, b_i \mid \sum U_j$	Lee et al. (1992)	Sung and Kim (2003)	Theorem 19
$Fm \mid p_{ij} = p_i, p\text{-batch}, b_i \mid \sum w_j U_j$	Baptiste (2000)	Theorem 19	Theorem 19

Results achieved in this paper are given in bold

To the best of our knowledge, each entry contains the first reference solving the corresponding problem or a generalization in polynomial time

have already been considered by Ackermann et al. (2020+). However, a close investigation of heuristics or approximation algorithms for PFBs in general is beyond the scope of this paper.

2 Optimality of permutation schedules

Our approach to scheduling a PFB relies on the well-known concept of permutation schedules. In a *permutation schedule* the order of the jobs is the same on all machines of the flow shop. This means there exists a permutation σ of the job indices such that $c_{i\sigma(1)} \leq c_{i\sigma(2)} \leq \dots \leq c_{i\sigma(n)}$, for all $i = 1, \dots, m$. Due to job-independent processing times and since a machine can only process one batch at a time, clearly the same then also holds for starting times instead of completion times. If there exists an optimal schedule S^* which is a permutation schedule with a certain ordering σ^* of the jobs, we say that *permutation schedules are optimal*. A job ordering σ^* which gives rise to an optimal permutation schedule is then called an *optimal job ordering*. Often, if the job ordering is clear, we will assume that jobs are already numbered with respect to that ordering and drop the notation of σ or σ^* . Finally, an ordering σ of the jobs is called an *earliest release date ordering*, if $r_{\sigma(1)} \leq r_{\sigma(2)} \leq \dots \leq r_{\sigma(n)}$.

Suppose that for some problem

$$Fm \mid r_j, p_{ij} = p_i, p\text{-batch}, b_i \mid f$$

permutation schedules are optimal. Then the scheduling problem can be split into two parts:

- (i) find an optimal ordering σ^* of the jobs;
- (ii) for each machine, partition the job set into batches and order those batches in accordance with the ordering σ^* , such that the resulting schedule is optimal.

In this section we deal with part (i) and show that for $f \in \{C_{\max}, \sum C_j\}$ optimal job orderings exist and can be

found in $O(n \log n)$ time. Then, in Sect. 3, we show that under certain, very general preconditions part (ii) can be done in polynomial time via dynamic programming.

We begin with a technical lemma which will help us to show optimality of permutation schedules in this section and also in Sect. 4.

Lemma 2 *Let S be a feasible schedule for a PFB and let σ be some earliest release date ordering of the jobs. Then there exists a feasible permutation schedule \hat{S} in which the jobs are ordered by σ and the multi-set of job completion times in \hat{S} is the same as in S .*

Proof Suppose that jobs are indexed according to ordering σ , i.e. σ is the identity permutation. Otherwise renumber jobs accordingly. Note that since σ is an earliest release date ordering, it follows that $r_1 \leq r_2 \leq \dots \leq r_n$. Let $B_\ell^{(i)}$, $\ell = 1, \dots, \kappa^{(i)}$, be the ℓ -th batch processed on M_i in S , where $\kappa^{(i)}$ is the total number of batches used on M_i . Let $s_\ell^{(i)}$ and $c_\ell^{(i)}$ be the start and completion times of batch $B_\ell^{(i)}$. Furthermore, let $k_\ell^{(i)} = |B_\ell^{(i)}|$ be the size of batch $B_\ell^{(i)}$ and let

$$j_\ell^{(i)} = \sum_{s=1}^{\ell} k_s^{(i)}$$

be the the number of jobs processed in batches $B_1^{(i)}, B_2^{(i)}, \dots, B_\ell^{(i)}$.

Construct the new schedule \hat{S} using batches $\hat{B}_\ell^{(i)}$, $i = 1, 2, \dots, m, \ell = 1, 2, \dots, \kappa^{(i)}$, with start times $\hat{s}_\ell^{(i)}$ and completion times $\hat{c}_\ell^{(i)}$ given by

$$\begin{aligned} \hat{B}_\ell^{(i)} &= \{J_{j_{\ell-1}^{(i)}+1}, J_{j_{\ell-1}^{(i)}+2}, \dots, J_{j_\ell^{(i)}}\}, \\ \hat{s}_\ell^{(i)} &= s_\ell^{(i)}, \\ \hat{c}_\ell^{(i)} &= c_\ell^{(i)}. \end{aligned}$$

In other words, in schedule \hat{S} we use batches with the same start and completion times and of the same size as in

S . We only reassign which job is processed in which batch in such a way, that the first batch on machine M_i processes the first $k_1^{(i)}$ jobs (in order of their numbering, i.e. in the earliest release date order), the second batch processes jobs $J_{k_1^{(i)}+1}$ to $J_{k_1^{(i)}+k_2^{(i)}}$, the third batch processes jobs $J_{k_1^{(i)}+k_2^{(i)}+1}$ to $J_{k_1^{(i)}+k_2^{(i)}+k_3^{(i)}}$ and so on. Such, in schedule \hat{S} , jobs are processed in order of their indices on all machines (i.e. in order σ).

Clearly, as all batches have exactly the same start and completion times, as well as the same sizes as in S , for schedule \hat{S} the multi-set of job completion times on the last machine is exactly the same as for S .

It remains to show that \hat{S} is feasible, i.e., no job is started on machine M_1 before its release date and no job is started on a machine M_i , $i = 2, 3, \dots, m$ before finishing processing on machine M_{i-1} .

We first show that no batch $\hat{B}_\ell^{(1)}$ on machine M_1 violates any release dates. Indeed, due to the feasibility of S , at the time $s_\ell^{(1)}$, when batch $B_\ell^{(1)}$ is started, at least $j_\ell^{(1)}$ jobs are released (recall that $j_\ell^{(1)}$ is the number of jobs processed in batches $B_1^{(1)}, B_2^{(1)}, \dots, B_\ell^{(1)}$). This means that in particular the first $j_\ell^{(1)}$ jobs in earliest release date order are released at time $s_\ell^{(1)}$. Thus, since batch $\hat{B}_\ell^{(1)}$ by construction contains only the jobs $J_{j_\ell^{(1)}+1}$ to $J_{j_\ell^{(1)}}$ and since jobs are numbered in earliest release date order, batch $\hat{B}_\ell^{(1)}$ does not violate any release dates.

Finally, using an analogous argument, we show that no batch starts processing on machine M_i before all jobs it contains finish processing on machine M_{i-1} , $i = 2, 3, \dots, m$. By time $s_\ell^{(i)}$, when batch $B_\ell^{(i)}$ is started on machine M_i in schedule S , machine M_{i-1} has processed at least $j_\ell^{(i)}$ jobs. Thus, by construction, the same is true for batch $\hat{B}_\ell^{(i)}$ in schedule \hat{S} . In particular, this means that at time $s_\ell^{(i)}$ the first $j_\ell^{(i)}$ jobs in order of their numbering are finished on machine M_{i-1} in schedule \hat{S} . Since batch $\hat{B}_\ell^{(i)}$ contains only the jobs $J_{j_\ell^{(i)}+1}$ to $J_{j_\ell^{(i)}}$, this means that batch $\hat{B}_\ell^{(i)}$ starts only after each job contained in it has finished processing on machine M_{i-1} . Therefore, schedule \hat{S} is feasible. \square

Note that, in particular, if all release dates are equal, then any ordering σ is an earliest release date ordering of the jobs. This fact will be used in Sect. 4.

Now we show that for a PFB with the makespan or total sum of completion times objective, permutation schedules are optimal and that any earliest release date ordering is an optimal ordering of the jobs. This result generalizes Sung and Yoon (1997), Proposition 1 to arbitrarily many machines. For makespan and total completion time, it also extends Sung et al. (2000), Lemma 1 to the case with release dates.

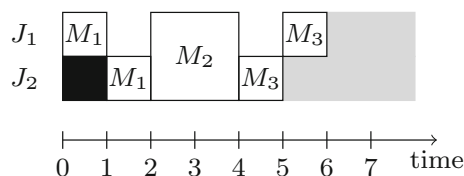


Fig. 3 A non-permutation schedule for the instance of Example 4

Theorem 3 For a PFB with objective function C_{\max} or $\sum C_j$, permutation schedules are optimal. Moreover, any earliest release date ordering is an optimal ordering of the jobs.

Proof Let S^* be an optimal schedule with respect to makespan or total completion time. Let σ be any earliest release date ordering. Using Lemma 2, construct a new permutation schedule \hat{S} with jobs ordered by σ on all machines and with the same multi-set of completion times as S . Since makespan and total completion time only depend on the multi-set of completion times, \hat{S} is optimal, too. \square

Hence, when minimizing the makespan or the total completion time in a PFB, it is valid to limit oneself to permutation schedules in an earliest release date order.

To conclude this section, we present an example where no permutation schedule is optimal for a bunch of other objective functions. This shows that Theorem 3 does not hold for these objective functions. It also implies that the statements made in Sung and Yoon (1997, Proposition 1), Sung et al. (2000, Lemma 1), and Sung and Kim (2003, Lemma 1) cannot be generalized to settings in which release dates and due dates or release dates and weights are present.

Example 4 Consider a PFB with $m = 3$ machines, $b_1 = p_1 = b_3 = p_3 = 1$, and $b_2 = p_2 = 2$. Suppose there are $n = 2$ jobs with release dates $r_1 = 0$ and $r_2 = 1$. Let due dates $d_1 = 6$ and $d_2 = 5$ be given. First, we show that no permutation schedule is optimal for L_{\max} . Figure 3 shows a job-oriented Gantt chart of a feasible schedule which is not a permutation schedule. Each rectangle labeled by a machine denotes a batch processed on this machine. The black box indicates that the second job has not been released yet. The gray shaded area denotes points in time after the due dates.

When constructing a permutation schedule for this instance, two decisions have to be made, namely which of the two jobs is processed first, and whether a batch of size two or two batches of size one should be used on M_2 . The resulting four possible schedules are depicted in Fig. 4.

However, the schedule in Fig. 3 achieves an objective value of zero, while all schedules in Fig. 4 have a positive objective value. Hence, there is no optimal permutation schedule for this instance. The same example shows that there is no optimal permutation schedule for $\sum T_j$ and $\sum U_j$.

Choosing weights $w_1 = 1$, $w_2 = 3$ leads to an instance where no permutation schedule is optimal for $\sum w_j C_j$, since

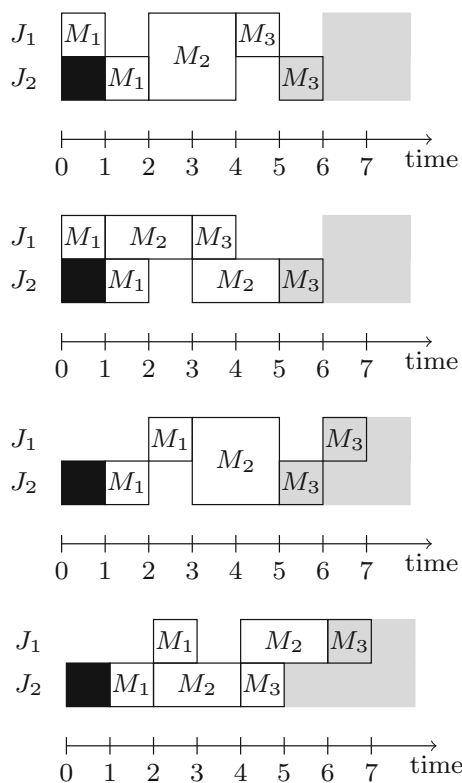


Fig. 4 All possible permutation schedules without unnecessary idle time for the instance of Example 4

the schedule in Fig. 3 achieves an objective value of $6 \cdot 1 + 5 \cdot 3 = 21$, while all schedules in Fig. 4 have an objective value of at least 22.

3 Dynamic programming to find an optimal schedule for a given job permutation

In this section we show that for a fixed number m of machines and a fixed ordering of the jobs σ , we can construct a polynomial time dynamic program, which finds a permutation schedule that is optimal among all schedules obeying job ordering σ . The algorithm works for an arbitrary regular sum or bottleneck objective function. Combined with Theorem 3 this shows that the makespan and the total completion time in a PFB can be minimized in polynomial time for fixed m .

The dynamic program is based on the important observation that, for a given machine M_i , the set of possible job completion times on M_i is not too large. In order to formalize this statement, we introduce the notion of a schedule being Γ -active. For ease of notation, we use the shorthand $[k] = \{1, 2, \dots, k\}$.

Definition 5 Given an instance of PFB with n jobs and m machines, let

$$\Gamma_i = \left\{ r_{j'} + \sum_{i'=1}^i \lambda_{i'} p_{i'} \mid j' \in [n], \lambda_{i'} \in [n] \text{ for } i' \in [i] \right\}$$

for all $i \in [m]$. We say that a schedule S is Γ -active, if $c_{ij}(S) \in \Gamma_i$, for any job $J_j, j \in [n]$, and any machine $M_i, i \in [m]$.

Note that on machine M_i , there are only $|\Gamma_i| \leq n^{i+1} \leq n^{m+1}$ possible job completion times to consider for any Γ -active schedule S .

Now we show that for a PFB problem with a regular objective function, any schedule can be transformed into a Γ -active schedule, without increasing the objective value. To do so, we prove that this is true even for a slightly stronger concept than Γ -active.

Definition 6 A schedule is called *batch-active*, if no batch can be started earlier without violating feasibility of the schedule and without changing the order of batches.

Clearly, given a schedule S that is not batch-active, by successively removing unnecessary idle times we can obtain a new, batch-active schedule S' . Furthermore, for regular objective functions, S' has objective value no higher than the original schedule S . These two observations immediately yield the following lemma.

Lemma 7 A schedule for a PFB can always be transformed into a batch-active schedule such that any regular objective function is not increased. This transformation does not change the order in which the jobs are processed on the machines. □

Now we show that, indeed, being batch-active is stronger than being Γ -active, in other words that every batch-active schedule is also Γ -active. This result generalizes an observation made by Baptiste (2000, Section 2.1) from a single machine to the flow shop setting.

Lemma 8 In a PFB, any batch-active schedule is also Γ -active.

Proof Fix some $i \in [m]$ and $j \in [n]$. Let $B_\ell^{(i)}$ be the batch which contains job J_j on machine M_i . Since the schedule is batch-active, $B_\ell^{(i)}$ is either started at the completion time of the previous batch $B_{\ell-1}^{(i)}$ on the same machine or as soon as all jobs of $B_\ell^{(i)}$ are available on machine M_i . In the former case, all jobs $J_{j'} \in B_\ell^{(i)}$ satisfy $c_{ij} = c_{ij'} + p_i$. In the latter case, there is a job $J_{j''} \in B_\ell^{(i)}$ such that $c_{ij} = c_{(i-1)j''} + p_i$, where we write $c_{0j''} = r_{j''}$ for convenience. The claim follows inductively by observing that the former case can happen at most $n - 1$ times in a row, since there are at most n batches on each machine. □

Together, Lemmas 7 and 8 imply the following desired property.

Lemma 9 *Let S be a schedule for a PFB problem with regular objective function f . Then there exists a schedule S' for the same PFB problem, such that*

1. *On each machine, jobs appear in the same order in S' as they do in S ,*
2. *S' has objective value no larger than S , and*
3. *S' is Γ -active.* □

In particular, Lemma 9 shows that, if for a PFB problem an optimal job ordering σ^* is given, then there exists an optimal schedule which is a permutation schedule with jobs ordered by σ^* and in addition Γ -active.

From this point on, we assume that the objective function is a regular sum or bottleneck function, that is, $f(C) = \bigoplus_{j=1}^n f_j(C_j)$, where $\bigoplus \in \{\sum, \max\}$ and f_j is nondecreasing for all $j \in [n]$. We also use the symbol \bigoplus as a binary operator. In what follows, we present a dynamic program which, given a job ordering σ , finds a schedule that is optimal among all Γ -active permutation schedules obeying job ordering σ . For simplicity, we assume that jobs are already indexed by σ . The dynamic program schedules the jobs one after the other, until all jobs are scheduled. Due to Lemma 9, if σ is an optimal ordering, then the resulting schedule is optimal.

Given an instance I of a PFB and a job index $j \in [n]$, let I_j be the modified instance which contains only the jobs J_1, \dots, J_j . We write $\Gamma = \Gamma_1 \times \Gamma_2 \times \dots \times \Gamma_m$ and $\mathcal{B} = [b_1] \times [b_2] \times \dots \times [b_m]$, where \times is the standard Cartesian product. For a vector

$$t = (t_1, t_2, \dots, t_i, \dots, t_{m-1}, t_m) \in \Gamma$$

of m possible completion times and a vector

$$k = (k_1, k_2, \dots, k_i, \dots, k_{m-1}, k_m) \in \mathcal{B}$$

of m possible batch sizes, we say that a schedule S for instance I_j corresponds to t and k if, for all $i \in [m]$, $c_{ij} = t_i$ and job J_j is contained in a batch with exactly k_i jobs (including J_j) on M_i .

Next, we define the variables g of the dynamic program. Let $\mathcal{S}(j, t, k)$ be the set of feasible permutation schedules S for instance I_j satisfying the following properties:

1. *Jobs in S are ordered by their indices,*
2. *S corresponds to t and k , and*
3. *S is Γ -active.*

Then, for $j \in [n]$, $t \in \Gamma$, and $k \in \mathcal{B}$, we define

$$g(j, t, k) = g(j, t_1, t_2, \dots, t_m, k_1, k_2, \dots, k_m)$$

to be the minimum objective value of a schedule in $\mathcal{S}(j, t, k)$ for instance I_j . If no such schedule exists, the value of g is defined to be $+\infty$.

Using the above definitions as well as Lemma 9, the objective value of the best permutation schedule ordered by the job indices is given by $\min_{t,k} g(n, t, k)$, $t \in \Gamma$, $k \in \mathcal{B}$. In Lemmas 10 and 11 (below) we provide formulas to recursively compute the values $g(j, t, k)$, $j = 1, 2, \dots, n$, $t \in \Gamma$, $k \in \mathcal{B}$. In total, we then obtain Algorithm 1 as our dynamic program.

Algorithm 1

Input: A PFB instance with regular sum or bottleneck objective function $f = \bigoplus_{j=1}^n f_j$.
Output: An optimal schedule among all permutation schedules ordered by the job indices.

- 1: For each $t \in \Gamma$ and $k \in \mathcal{B}$, compute $g(1, t, k)$ via the formula defined in Lemma 10.
- 2: For each $j = 2, 3, \dots, n$, $t \in \Gamma$, and $k \in \mathcal{B}$, compute $g(j, t, k)$ via the formula defined in Lemma 11. If $g(j, t, k) < +\infty$, also store a reference to the minimizers t' and k' on the right-hand side of the recurrence in Lemma 11.
- 3: Find t and k minimizing $g(n, t, k)$ and start from them to backtrack the references stored in Step 2 in order to find the optimal schedule.

The remainder of this section deals with proving the correctness and running time bound of Algorithm 1. As mentioned above, in Lemmas 10 and 11 we define and prove the correctness of a recursive formula to compute all values of function g . Lemma 10 deals with the starting values $g(1, t, k)$ while Lemma 11 deals with the recurrence relation. Finally, in Theorem 12 we connect the results of this section to prove the correctness of Algorithm 1.

Lemma 10 *For $j = 1$, $t_i \in \Gamma_i$, and $k_i \in [b_i]$, $i \in [m]$, the starting values of g are given by*

$$g(1, t, k) = \begin{cases} f_1(t_m), & \text{if conditions (i)–(iii) hold,} \\ +\infty, & \text{otherwise,} \end{cases}$$

where

- (i) $k_i = 1$ for all $i \in [m]$,
- (ii) $t_1 \geq r_1 + p_1$, and
- (iii) $t_{i+1} \geq t_i + p_{i+1}$ for all $i \in [m - 1]$.

Proof Conditions (i)–(iii) are necessary for the existence of a schedule in $\mathcal{S}(1, t, k)$ because I_1 consists of only one job and there must be enough time to process this job on each machine. Conversely, if (i)–(iii) are satisfied, then the vector t of completion times uniquely defines a feasible schedule

$S \in \mathcal{S}(1, t, k)$ by $c_{i1}(S) = t_i$. Since S is uniquely defined by t and thus the only schedule in $\mathcal{S}(1, t, k)$, its objective value $f_1(C_1) = f_1(c_{m1}) = f_1(t_m)$ is optimal. \square

Now we turn to the recurrence formula to calculate g for $j > 1$ from the values of g for $j - 1$.

Lemma 11 For $j > 1$, $t_i \in \Gamma_i$, and $k_i \in [b_i]$, $i \in [m]$, the values of g are determined by

$$g(j, t, k) = \begin{cases} \min\{f_j(t_m) \oplus g(j - 1, t', k') \mid (**)\}, & \text{if } (*), \\ +\infty, & \text{otherwise,} \end{cases}$$

where the minimum over the empty set is defined to be $+\infty$. Here, $(*)$ is given by conditions

- (i) $t_1 \geq r_j + p_1$ and
- (ii) $t_{i+1} \geq t_i + p_{i+1}$ for all $i \in [m - 1]$,

and $(**)$ is given by conditions

- (iii) $t'_i \in \Gamma_i$,
- (iv) $k'_i \in [b_i]$,
- (v) if $k_i = 1$, then $t'_i \leq t_i - p_i$, and
- (vi) if $k_i > 1$, then $t'_i = t_i$ and $k'_i = k_i - 1$,

for all $i \in [m]$.

Proof Fix values $t_i \in \Gamma_i$ and $k_i \in [b_i]$, $i \in [m]$. The conditions of $(*)$ are necessary for the existence of a schedule in $\mathcal{S}(j, t, k)$ because there must be enough time to process job J_j on each machine. Therefore, g takes the value $+\infty$, if $(*)$ is violated. For the remainder of the proof, assume that $(*)$ is satisfied. Hence, we have to show that

$$g(j, t, k) = \min \{ f_j(t_m) \oplus g(j - 1, t', k') \mid (**)\}. \tag{3}$$

We first prove “ \geq ”. If the left-hand side of (3) equals infinity, then this direction follows immediately. Otherwise, by the definition of g , there must be a schedule $S \in \mathcal{S}(j, t, k)$ with objective value $g(j, t, k)$. Schedule S naturally defines a feasible schedule S' for instance I_{j-1} by ignoring job J_j .

Observe that, because S belongs to $\mathcal{S}(j, t, k)$ and therefore is Γ -active, S' is also Γ -active and job J_{j-1} finishes processing on machine M_i at some time $t'_i \in \Gamma_i$. Also, since S is feasible, on each machine M_i job J_{j-1} is scheduled in some batch of size $k'_i \leq b_i$. Thus, S' corresponds to two unique vectors $t' = (t'_1, t'_2, \dots, t'_m)$ and $k' = (k'_1, k'_2, \dots, k'_m)$, which satisfy (iii) and (iv) from $(**)$. Note that, in particular, $S' \in \mathcal{S}(j - 1, t', k')$.

Furthermore, t' and k' also satisfy (v) and (vi) from $(**)$. Indeed, due to the fixed job permutation, one of the following two things happens on each machine M_i in S : either jobs

J_j and J_{j-1} are batched together, or job J_j is batched in a singleton batch. In the former case, it follows that $1 < k_i = k'_i + 1$ and $t'_i = t_i$, while the latter case requires $k_i = 1$ and $t_i \geq t'_i + p_i$, since the machine is occupied by the previous batch with job J_{j-1} until then.

Thus, t' and k' satisfy $(**)$ and we obtain

$$\begin{aligned} g(j, t, k) &= \bigoplus_{j'=1}^j f_{j'}(C_{j'}(S)) \\ &= f_j(t_m) \oplus \bigoplus_{j'=1}^{j-1} f_{j'}(C_{j'}(S')) \\ &\geq f_j(t_m) \oplus g(j - 1, t', k'), \end{aligned}$$

where the last inequality follows due to the definition of g and $S' \in \mathcal{S}(j - 1, t', k')$. Hence, the “ \geq ” direction in (3) follows because t' and k' satisfy $(**)$.

For the “ \leq ” direction, if the right-hand side of (3) equals infinity, then this direction follows immediately. Otherwise, let t' and k' be minimizers at the right-hand side. By definition of g there must be a schedule $S' \in \mathcal{S}(j - 1, t', k')$ for I_{j-1} with objective value $g(j - 1, t', k')$.

We now show that S' can be extended to a feasible schedule $S \in \mathcal{S}(j, t, k)$. Construct S from S' by adding job J_j in the following way:

- if in S' there is a batch on machine M_i which ends at time t_i , add J_j to that batch (we show later that this does not cause the batch to exceed the maximum batch size b_i of M_i);
- otherwise, add a new batch on machine M_i finishing at time t_i and containing only job J_j (we show later that this does not create overlap with any other batch on machine M_i).

First note that $c_{ij}(S) = t_i$ for all $i \in [m]$, and thus S corresponds to t by definition. Furthermore, for each machine M_i consider the two cases (a) $k_i = 1$ and (b) $k_i > 1$.

In case (a), due to (v) it follows that $t'_i \leq t_i - p_i$. Since job J_{j-1} finishing at time t'_i is the last job completed on machine M_i in schedule S' , by construction in schedule S job J_j is in a singleton batch on machine M_i that starts at time $t_i - p_i$ and ends at time t_i . Therefore, in this case, S corresponds to k , because $k_i = 1$ and job J_j is in a singleton batch. Also, since the last batch on machine M_i in S' ends at time $t'_i \leq t_i - p_i$ and the batch with job J_j starts at time $t_i - p_i$, no overlapping happens between the new batch for job J_j and any other batch on machine M_i .

In case (b), due to (vi) it follows that $t'_i = t_i$ and by construction job J_j is scheduled in the same batch as job J_{j-1} on machine M_i in S . Since S' corresponds to k' and again due to (vi), this means that J_j is scheduled in a batch with

k_i jobs on machine M_i and S corresponds to k in this case. Also, since $k_i \in [b_i]$ by definition, no batch in S exceeds its permissible size.

Combining the considerations for cases (a) and (b), together with the feasibility of schedule S' , it follows that

- (1) S corresponds to k ,
- (2) no overlapping happens between batches in S , and
- (3) all batches in S are of permissible size.

In order to show feasibility of S it is only left to show that no job starts before its release date on machine M_1 , and no job starts on machine M_i before it is completed on machine M_{i-1} . As S' is feasible, this is clear for all jobs other than J_j . On the other hand, since S corresponds to t , and t fulfills (*), it also follows for job J_j .

Thus, S is indeed feasible. In total, we have shown all conditions for $S \in \mathcal{S}(j, t, k)$. Therefore, we obtain

$$\begin{aligned} g(j, t, k) &\leq \bigoplus_{j'=1}^j f_{j'}(C_{j'}(S)) \\ &= f_j(t_m) \oplus \bigoplus_{j'=1}^{j-1} f_{j'}(C_{j'}(S')) \\ &= f_j(t_m) \oplus g(j-1, t', k'), \end{aligned}$$

where the last equality is due to the choice of S' as a schedule with objective value $g(j-1, t', k')$. Hence, the “ \leq ” direction in (3) follows due to the choice of t' and k' as minimizers of the right-hand side. \square

Using Lemmas 9, 10, and 11 we can now prove correctness of Algorithm 1 and show that its runtime is bounded polynomially (if m is fixed).

Theorem 12 Consider a PFB instance with a constant number of m machines and a regular sum or bottleneck objective function. Then, for a given ordering of the jobs, Algorithm 1 finds the best permutation schedule in time $\mathcal{O}(n^{m^2+4m+1})$.

Proof Without loss of generality, assume that the desired job ordering is given by the job indices. By Lemmas 10 and 11, Algorithm 1 correctly computes the values of function g . Lemma 9 assures that the resulting schedule is indeed optimal among all permutation schedules ordered by the job indices.

Concerning the time complexity, note that there are $|\Gamma|$ many possible choices for vector t and $|\mathcal{B}|$ many possible choices for vector k , yielding a total of $|\Gamma| \cdot |\mathcal{B}|$ possible combinations for vectors t and k . The number can be bounded

from above as follows:

$$\begin{aligned} |\Gamma| \cdot |\mathcal{B}| &= |\Gamma_1| \cdot |\Gamma_2| \cdot \dots \cdot |\Gamma_m| \cdot b_1 \cdot b_2 \cdot \dots \cdot b_m \\ &\leq n^2 \cdot n^3 \cdot \dots \cdot n^{m+1} \cdot n^m \\ &= n^{\frac{(m+1)(m+2)}{2} - 1 + m} \\ &= n^{\frac{m^2}{2} + \frac{5m}{2}}. \end{aligned}$$

Here, the first m factors in the second line are due to $\Gamma_i \leq n^{i+1}$ and the last factor n^m is due to inequalities $b_i \leq n$ for all $i \in [m]$.

Therefore, Step 1 of Algorithm 1 involves at most $n^{\frac{m^2}{2} + \frac{5m}{2}}$ iterations, each of which can be performed in constant time because m is fixed.

For Step 2, first observe that conditions (*) and (**) can be checked in constant time for fixed m . In a single iteration, that is, for fixed j, t , and k , in order to compute the minimum in the recurrence of Lemma 11, we may have to consider all possible choices for t' and k' . The number of these choices is $|\{(t', k') \in \Gamma \times \mathcal{B} \mid (**)\}|$. Aggregating over all t and k , while keeping j fixed, yields

$$\begin{aligned} &\sum_{(t,k) \in \Gamma \times \mathcal{B}} |\{(t', k') \in \Gamma \times \mathcal{B} \mid (**)\}| \\ &= |\{(t, t', k, k') \in \Gamma \times \Gamma \times \mathcal{B} \times \mathcal{B} \mid (**)\}| \\ &= \prod_{i=1}^m |\{(t_i, t'_i, k_i, k'_i) \in \Gamma_i \times \Gamma_i \times [b_i] \times [b_i] \\ &\quad \mid (v) \text{ and } (vi)\}|. \end{aligned}$$

The number of quadruples (t, t', k, k') satisfying (v) and (vi) can be bounded by $b_i |\Gamma_i|^2 + b_i |\Gamma_i|$. Here, the first term corresponds to the quadruples with $k_i = 1$, while the second term corresponds to the quadruples with $k_i > 1$, in which case t'_i and k'_i are uniquely determined by t_i and k_i . Further, note that due to $|\Gamma_i| \geq 1$, it holds that $b_i |\Gamma_i|^2 + b_i |\Gamma_i| \leq 2b_i |\Gamma_i|^2$. Hence, the total running time of Step 2 (also aggregating over all choices of j), is at most

$$\begin{aligned} &\mathcal{O}\left(n \prod_{i=1}^m 2b_i |\Gamma_i|^2\right) \\ &= \mathcal{O}\left(n \prod_{i=1}^m b_i |\Gamma_i|^2\right) \\ &\subseteq \mathcal{O}\left(n \prod_{i=1}^m n \cdot n^{2i+2}\right) \\ &= \mathcal{O}(n^{m^2+4m+1}). \end{aligned} \tag{4}$$

The first equality is due to the factor 2^m being constant for constant m and the set inclusion is due to $b_i \leq n$ and $|\Gamma_i| \leq n^{i+1}$.

Finally, in Step 3, we take the minimum over at most $n^{\frac{m^2}{2} + \frac{5m}{2}}$ many values. The backtracking and reconstruction of the final schedule can be done in time $\mathcal{O}(n)$ for constant m .

Hence, in total, Step 2 is the bottleneck and the total time complexity is $\mathcal{O}(n^{m^2+4m+1})$. \square

Combining Theorems 3 and 12, we obtain:

Corollary 13 *For a PFB with release dates and a fixed number m of machines, the makespan can be minimized in $\mathcal{O}(n^{m^2+4m+1})$ time. The same holds for the total completion time.* \square

In particular, for any fixed number m of machines, the problems

$$Fm \mid r_j, p_{ij} = p_i, p\text{-batch}, b_i \mid f$$

with $f \in \{C_{\max}, \sum C_j\}$ can be solved in polynomial time.

3.1 Improved running time for makespan minimization

Sung et al. (2000, Theorem 6) showed that for minimizing the makespan in a PFB, it is optimal to use the so-called first-only-empty batching on the last machine M_m , that is, to completely fill all batches on the last machine, except for the first batch, which might contain less jobs if n is not a multiple of b_m . In a permutation schedule ordered by the job indices with first-only-empty batching on M_m , a makespan of c can be achieved if and only if for each job J_j we have enough time to process the $n - j + 1$ jobs J_j, J_{j+1}, \dots, J_n on M_m after J_j has been completed on M_{m-1} . In other words, a makespan of c can be achieved, if and only if $c \geq c_{(m-1)j} + \left\lceil \frac{n-j+1}{b_m} \right\rceil$ for every $j \in [n]$. Hence, the optimal makespan is given by $\max_{j=1,2,\dots,n} c_{(m-1)j} + \left\lceil \frac{n-j+1}{b_m} \right\rceil$. Therefore, the problem to minimize the makespan on an m -machine PFB can be solved in the following way: We first find a schedule for the first $m - 1$ machines that minimizes the objective function $\bigoplus_{j=1}^n f_j(c_{(m-1)j})$ with $\bigoplus = \max$ and $f_j(c_{(m-1)j}) = c_{(m-1)j} + \left\lceil \frac{n-j+1}{b_m} \right\rceil$. By Theorem 12, this can be done in time

$$\mathcal{O}(n^{(m-1)^2+4(m-1)+1}) = \mathcal{O}(n^{m^2+2m-2})$$

for fixed m . Afterwards, this schedule is extended by a first-only-empty batching on M_m , which does not further increase the asymptotic runtime. Thus, for the makespan objective, we can strengthen Corollary 13:

Corollary 14 *For a PFB with release dates and a fixed number m of machines, the makespan can be minimized in $\mathcal{O}(n^{m^2+2m-2})$ time.* \square

4 Equal release dates

The dynamic program presented in the previous section works for a very general class of objective functions. Still, Corollary 13 only holds for makespan and total completion time because for other standard objective functions permutation schedules are not optimal, as shown in Example 4.

However, in the case where all release dates are equal, it turns out that permutation schedules are always optimal. We first show that in the case of equal release dates, any feasible schedule S can be transformed into a feasible permutation schedule S' with equal objective value.

Lemma 15 *Let S be a feasible schedule for a PFB where all jobs are released at time 0. Then there exists a feasible permutation schedule S' such that $C_j(S) = C_j(S')$ for all $j = 1, 2, \dots, n$.*

Proof Let σ be the ordering of jobs on the last machine. Since $r_j = 0$ for all $j = 1, 2, \dots, n$, ordering σ is an earliest release date ordering. Thus we can use Lemma 2 to construct a permutation schedule S' , where jobs are ordered by σ on all machines and the multi-set of completion times is the same as for S . Furthermore, since the ordering on the last machine in S' is the same as in S , in S' each job has the same completion time as in S , i.e. $C_j(S) = C_j(S')$ for all $j = 1, 2, \dots, n$. \square

From Lemma 15, it follows that permutation schedules are optimal for PFBs with equal release dates, as stated in the following theorem. The theorem generalizes the result presented by Rachamadugu et al. (1982) for usual proportionate flow shop without batching. Also compare Sung et al. (2000, Lemma 1). In the theorem, we use the term *scheduling objective function* to describe any objective function only dependent on the completion times of the jobs.

Theorem 16 *For any scheduling objective function, if an optimal schedule exists for a PFB problem without release dates, then there also exists an optimal permutation schedule.*

Proof Apply Lemma 15 to an optimal schedule S^* in order to obtain a permutation schedule S' . Note that, by Lemma 15, for every job, the completion time on the last machine in schedule S' is exactly the same as in schedule S^* . Thus, for any completion time dependent objective function, schedules S^* and S' have the same objective value. \square

Note that for all regular objective functions (including those studied in this paper), at least one optimal solution always exists. Indeed, Lemma 7 shows that for regular objective functions batch-active schedules are optimal, and there are only finitely many batch active schedules for a PFB. Thus, the restriction in the statement of the theorem is only a formality. However, Theorem 16 holds for a much wider range of scheduling objective functions than studied in this paper.

Importantly, since finish times for all jobs stay exactly the same when moving from schedule S^* to schedule S' in the proof of Theorem 16, the theorem holds also for non-regular objective functions (as long as they are defined in such a way that at least one optimal solution exists).

Of course, it may still be hard to find optimal permutations. However, the following theorem shows that for several traditional objective functions, an optimal permutation can be found efficiently. For the solution of due date related objectives, earliest due date orderings are of particular importance. They are defined analogously to earliest release date orderings: an ordering σ is an *earliest due date ordering*, if $d_{\sigma(1)} \leq d_{\sigma(2)} \leq \dots \leq d_{\sigma(n)}$.

Theorem 17 Consider a PFB without release dates.

- (i) For minimizing the total weighted completion time, any ordering by non-increasing weights is optimal.
- (ii) For minimizing maximum lateness and total tardiness, any earliest due date ordering is optimal.

Proof For proving (i), assume jobs are indexed by any given non-increasing order of weights, i.e., $w_1 \geq w_2 \geq \dots \geq w_n$. By Theorem 16, there exists an optimal permutation schedule S^* minimizing the total weighted completion time. Suppose there is at least one pair of jobs J_{j_1}, J_{j_2} with $j_1 < j_2$, but $C_{j_1}(S^*) > C_{j_2}(S^*)$. Swapping J_{j_1} and J_{j_2} in S^* yields a new permutation schedule S' , with objective value

$$\sum w_j C_j(S') = \sum w_j C_j(S^*) + (w_{j_1} - w_{j_2})(C_{j_2}(S^*) - C_{j_1}(S^*)).$$

Due to $j_1 < j_2$, we have $w_{j_1} \geq w_{j_2}$ and with $C_{j_1}(S^*) > C_{j_2}(S^*)$ it follows that

$$(w_{j_1} - w_{j_2})(C_{j_2}(S^*) - C_{j_1}(S^*)) \leq 0.$$

Therefore, it we obtain

$$\sum w_j C_j(S') \leq \sum w_j C_j(S^*).$$

Performing such exchanges of jobs sequentially, we eventually obtain an optimal permutation schedule in which jobs are scheduled in the given order of non-increasing weights. Hence, any given ordering of the jobs by non-increasing weights is optimal.

Part (ii) can be proven by exchange arguments analogous to (i). A detailed proof can be found in the appendix. \square

From Theorems 12 and 17, we obtain the following complexity results.

Theorem 18 For a PFB without release dates and a fixed number m of machines, the total weighted completion time

can be minimized in $O(n^{m^2+2m-2})$ time. The same holds for the maximum lateness and the total tardiness.

Proof With a time complexity of $O(n^{m^2+4m+1})$, this statement follows directly from Theorems 12 and 17. The improvement in the running time can be obtained by the following two observations. First note that, due to vanishing release dates, the size of Γ_i is at most n^i instead of n^{i+1} . Comparing with the estimate (4) in the proof of Theorem 12, this yields an improvement by the factor n^{2m} because each of the m factors $|\Gamma_i|^2$ can be bounded by n^{2i} instead of n^{2i+2} . Second, Sung et al. (2000, Theorem 1) showed that it is optimal to use the so-called last-only-empty batching on the first machine M_1 , that is, to completely fill all batches on M_1 , except for the last batch, which might contain less jobs if n is not a multiple of b_1 . If we do so, the schedule of M_1 is fixed. This also fixes all values of the parameters k_1 and t_1 dependent on the current job index j . Hence, in the runtime estimation (4), we may leave out the factor $b_1 |\Gamma_1|^2$. Since both Γ_1 and b_1 are bounded by n (already considering the improvement above), this results in a further improvement by a factor of n^3 in the total runtime. Therefore, we obtain a total time complexity of

$$O(n^{m^2+4m+1-2m-3}) = O(n^{m^2+2m-2}). \quad \square$$

It is also possible to minimize the (weighted) number of late jobs in $O(n^{m^2+2m-1})$ time. However, to achieve this result, it is necessary to adjust the algorithm from Sect. 3 slightly. Suppose that jobs are ordered in an earliest due date order and suppose further \mathcal{J} is the set of on-time jobs in an optimal solution for PFB to minimize the (weighted) number of late jobs. Note that in this case, we can find an optimal schedule by first scheduling all jobs in \mathcal{J} , in earliest due date order, using the algorithm from Sect. 3, and then schedule all late jobs in an arbitrary, feasible way.

Usually, however, the optimal set of on-time jobs is not known in advance. Thus the dynamic program has to be adapted in such a way that it finds the optimal set of on-time jobs and a corresponding optimal schedule simultaneously. This can be done by, roughly speaking, allowing the algorithm to ignore a job: in the j -th step of the algorithm, when job J_j (in earliest due date order) is scheduled, the algorithm can not only decide to schedule the job in sequence, but can alternatively decide to skip it, making it late (and paying a penalty w_j). In all other regards, the algorithm remains exactly the same as presented in Sect. 3. The technical details to prove correctness and running time of the algorithm are, as one would expect, very similar to Sect. 3 and are moved to the appendix.

We obtain the following theorem. The additional factor n in comparison to Theorem 18 stems from the fact that we do not know the optimal permutation in advance, and therefore

we cannot make use of last-only-empty batching on the first machine as easily as in the proof of Theorem 18.

Theorem 19 *For a PFB without release dates and a fixed number m of machines, the weighted number of late jobs can be minimized in $O(n^{m^2+2m-1})$ time.*

In particular, combining Theorems 18 and 19, we obtain that, for any fixed number m of machines, the problems

$$Fm \mid p_{ij} = p_i, p\text{-batch}, b_i \mid f$$

with $f \in \{\sum w_j C_j, L_{\max}, \sum T_j, \sum U_j, \sum w_j U_j\}$ can be solved in polynomial time.

Note that, while minimizing makespan and total completion time in a PFB with a fixed number m of machines and without release dates can be handled in the same manner, a time complexity of $O(n^{m^2+2m-2})$ is strictly speaking no longer polynomial for those two problems. Indeed, since in problems

$$Fm \mid p_{ij} = p_i, p\text{-batch}, b_i \mid f$$

with $f \in \{C_{\max}, \sum C_j\}$, jobs are completely identical, an instance can be encoded concisely by simply encoding the number n of jobs, rather than every job on its own. Such an encoding of the jobs takes only $O(\log n)$ space, meaning that even algorithms linear in n would no longer be polynomial in the length of the instance.

In the literature, scheduling problems for which such a concise encoding in $O(\log n)$ space is possible are usually referred to as *high-multiplicity scheduling problems* (Hochbaum and Shamir 1990, 1991). They are often difficult to handle and require additional algorithmic consideration, since even specifying a full schedule normally takes at least $O(mn)$ time (one start and end time for each job on each machine). In order to resolve this issue, for these high-multiplicity scheduling problems special attention is often paid to finding optimal schedules with specific, regular structures. Such a structure may then allow for a concise $O(\log n)$ encoding of the solution. Notice, however, that a priori the dynamic programming algorithm presented in this paper does not give rise to such a regular structure.

In the case of $m = 2$ machines and the makespan objective, Hertrich (2018) shows that the $O(n)$ algorithm by Ahmadi et al. (1992) can be adjusted to run in polynomial time, even in the high multiplicity sense. For all other cases, the theoretical complexity of the two problems

$$Fm \mid p_{ij} = p_i, p\text{-batch}, b_i \mid f$$

with $f \in \{C_{\max}, \sum C_j\}$ is left open for future research. However, note that in practical applications jobs usually have additional meaning attached (for instance patient identifiers,

in our pharmacy example from the beginning) and thus such instances are usually encoded with a length of at least n .

5 Conclusions and future work

In this paper, motivated by an application in modern pharmaceutical production, we investigated the complexity of scheduling a proportionate flow shop of batching machines (PFB). The focus was to provide the first complexity results and polynomial time algorithms for PFBs with $m > 2$ machines.

Our main result is the construction of a new algorithm, using a dynamic programming approach, which schedules a PFB with release dates to minimize the makespan or total completion time in polynomial time for any fixed number m of machines. In addition, we showed that, if all release dates are equal, the constructed algorithm can also be used to minimize the weighted total completion time, the maximum lateness, the total tardiness, and the (weighted) number of late jobs. Hence, for a PFB without release dates and with a fixed number of m machines, these objective functions can be minimized in polynomial time. Previously these complexity results were only known for the special case of $m = 2$ machines, while for each $m \geq 3$ the complexity status was unknown. For minimizing weighted total completion time and weighted number of late jobs, these results were even unknown in the case of $m = 2$.

An important structural result in this paper is that permutation schedules are optimal for PFBs with release dates and the makespan or total completion time objective, as well as for any PFB problem without release dates. This result was needed in order to show that the constructed algorithm can be correctly applied to the problems named above.

Concerning PFBs with release dates, recall that in the presence of release dates permutation schedules are not necessarily optimal for traditional scheduling objective functions other than makespan and total completion time (see Example 4). However, there are several special cases for which permutation schedules remain optimal, in particular if the order of release dates corresponds well to the order of due dates (in the case of due date objectives) and/or weights (in the case of weighted objectives). For example, Hertrich (2018) shows that, when minimizing the weighted total tardiness, if there exists a job order σ that is an earliest release date ordering, an earliest due date ordering, and a non-increasing weight ordering at the same time, then there exists an optimal schedule which is a permutation schedule with jobs ordered by σ . This, of course, implies analogous results for weighted total completion time, total tardiness and maximum lateness (again, see Hertrich (2018)). In those special cases, clearly the algorithm presented in this paper can be used to solve the problems in polynomial time. Still, the complexity status of

PFBs with release dates in the general case, where orderings do not correspond well with each other, remains open for all traditional scheduling objectives other than makespan and total completion time.

Another problem left open for future study is the complexity of minimizing the weighted total tardiness in the case where all release dates are equal. This problem is open, even for the special case of $m = 2$. In this paper, it is shown that permutation schedules are optimal for all objective functions in the absence of release dates. Furthermore, if an optimal job order is known and the number of machines is fixed, then an optimal schedule can be found in polynomial time, using the algorithm presented in this paper.

Unfortunately, the complexity of finding an optimal job order to minimize the weighted total tardiness in a PFB remains open. Note that for scheduling a single batching machine with equal processing times and for usual proportionate flow shop without batching the weighted total tardiness can be minimized in polynomial time (Brucker and Knust (2009)). Here, proportionate flow shop can be reduced to the single machine problem with all processing times equal to the processing time of the bottleneck machine (see, e.g., Sung et al. (2000)). However, such a reduction is not possible for PFBs, since there may be several local bottleneck machines which influence the quality of a solution.

Observe also that, in the special case of $m = 2$ machines, given an arbitrary schedule for the first machine, finding an optimal schedule for the second machine is similar to solving problem $1 \mid p_j = p, r_j \mid \sum w_j T_j$ or the analogous problem on a batching machine, which also both have open complexity status (see Baptiste (2000), Brucker and Knust (2009)). So attempting to split up the problem in such a manner does not work either. Of course, if instead of an arbitrary schedule on the first machine, some special schedule is selected, it may be possible that the special structure for the release dates on the second machine helps to solve the problem. Notice, though, that this still leaves open the question of which schedule to select on the first machine.

The last open complexity question concerns the case where the number of machines is no longer fixed, but instead part of the input. We are unfortunately not aware of any promising approaches to find reductions from NP-hard problems. However, interestingly, Hertrich (2018) proves that for all versions of PFBs, minimizing the total completion time is always at least as hard as minimizing the makespan. This reduction does not hold for scheduling problems in general.

One noteworthy special case, where positive complexity may be more readily achievable, arises from fixing the number of jobs n , leaving only the number of machines m as part of the input. Note that in this case, high-multiplicity, as described in Sect. 4, is not a concern: since n is fixed, a schedule can be put out in $O(m)$ time and since each machine has an individual processing time, the input also has at least

size $O(m)$. For PFBs with exactly two jobs, $n = 2$, and an arbitrary number of machines, Hertrich (2018) shows that the makespan can be minimized in $O(m^2)$ time. A machine-wise dynamic program is provided that finds all schedules, in which the completion time of one job cannot be improved without delaying the other job. In other words, the program constructs all Pareto-optimal schedules, if the completion time of each job is considered as a separate objective. If the number of jobs n is larger than 2, then the dynamic program can still be applied, but its runtime in that case is pseudo-polynomial (see Hertrich (2018)).

Finally, note that while the algorithm provided in this paper has the benefit of being very general, thus being applicable to many different problems, it incurs some practical limitations due to its relatively large running time. For example, for $m = 2$, our algorithm to minimize the makespan has a runtime of $O(n^6)$ (Corollary 14). In contrast, the algorithm specialized to the case of $m = 2$ by Sung and Yoon (1997) runs in $O(n^2)$. The reason for the larger runtime of our dynamic program is that, due to its generality, it needs to consider all possible job completion times in the set Γ , while the algorithm by Sung and Yoon (1997) makes use of the fact that this is not necessary in the special case $m = 2$.

Now that many of the previously open complexity questions are solved, for future research it would be interesting to find more practically efficient algorithms, in particular for small numbers of machines like $m = 3$ or $m = 4$, and/or for specific objective functions. This might be possible by using some easy structural properties, which were not of use for speeding up the algorithm in the general case but which may well help with dedicated algorithms for small numbers of machines. For example, if the objective function is regular, then on any machine, when a batch is started, it is always optimal to include as many jobs as possible (either all waiting jobs, or as many as the maximum batch size). For the general algorithm provided in this paper, this property was not of use, as our algorithm adds the jobs one by one to the schedule and it only becomes clear how many jobs wait at a machine during the backtracking phase. For a dedicated algorithm, on the other hand, it may well be possible use this property to reduce the number of possible schedules that have to be considered.

If fast, dedicated algorithms cannot be obtained, then instead approximations or heuristics need to be considered in order to solve PFBs in practice. In the literature, approximations and heuristics often use restrictions to permutation schedules in order to simplify difficult scheduling problems. To this end, the results on the optimality of permutation schedules presented in this paper can be of great use to future work. In particular, the results from Sect. 2 can be of help when designing fast heuristics for makespan, total completion time and related objective functions. For the same reason, it would be interesting to quantify the quality of permutation

schedules (e.g. in terms of an approximation factor) for PFB problems where permutation schedules are not optimal.

Another approach to improving the algorithms’ efficiency, especially if negative complexity results for the case with an arbitrary number of machines are achieved, would be to consider PFBs from a parameterized complexity point of view. Recently, parameterized complexity has started to receive more attention in the scheduling community (cf. Mnich and Wiese (2015), Weiß et al. (2016), Mnich and van Bevern (2018), Diessel and Ackermann (2019)). A problem with input size n is said to be *fixed-parameter tractable* with respect to a parameter k if it can be solved in a running time of $\mathcal{O}(f(k)p(n))$ where f is an arbitrary computable function and p a polynomial not depending on k . Although our algorithm is polynomial for any fixed number m of machines, it is not fixed-parameter tractable in m because m appears in the exponent of the running time. Clearly, a fixed-parameter tractable algorithm would be preferable, both in theory and, most likely, in practice, if one could be found.

Funding Open Access funding provided by Projekt DEAL. The research of Christoph Hertrich was supported by the Felix-Klein-Zentrum für Mathematik e.V. and by the DFK-GRK 2434 Facets of Complexity.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

Appendix A: Proof of Theorem 17 (ii)

In this appendix we give a detailed proof of Theorem 17 (ii). Assume jobs are indexed in any given earliest due date ordering, i.e. $d_1 \leq d_2 \leq \dots \leq d_n$. By Theorem 16, there exists an optimal permutation schedule S^* with respect to maximum lateness or total tardiness. Suppose there is at least one pair of jobs J_{j_1}, J_{j_2} with $j_1 < j_2$, but $C_{j_1}(S^*) > C_{j_2}(S^*)$. Swapping J_{j_1} and J_{j_2} in S^* yields a new permutation schedule S' .

We first show that $L_{\max}(S') \leq L_{\max}(S^*)$. Because the lateness of all jobs other than J_{j_1} and J_{j_2} remains unchanged when performing the swap, it suffices to show that

$$\max\{L_{j_1}(S'), L_{j_2}(S')\} \leq \max\{L_{j_1}(S^*), L_{j_2}(S^*)\}$$

where $L_j(S') = C_j(S') - d_j$. However, due to $j_1 < j_2$, we have $d_{j_1} \leq d_{j_2}$. Hence, with $C_{j_1}(S^*) > C_{j_2}(S^*)$ it follows that

$$L_{j_1}(S') = C_{j_2}(S^*) - d_{j_1} < C_{j_1}(S^*) - d_{j_1} = L_{j_1}(S^*)$$

and

$$L_{j_2}(S') = C_{j_1}(S^*) - d_{j_2} \leq C_{j_1}(S^*) - d_{j_1} = L_{j_1}(S^*),$$

which finishes the proof for maximum lateness.

Finally, we show that $\sum T_j(S') \leq \sum T_j(S^*)$. Because the tardiness of all jobs other than J_{j_1} and J_{j_2} remains unchanged when performing the swap, it suffices to show that

$$T_{j_1}(S') + T_{j_2}(S') \leq T_{j_1}(S^*) + T_{j_2}(S^*)$$

where $T_j(S') = \max\{C_j(S') - d_j, 0\}$. We distinguish three cases. Firstly, suppose J_{j_1} and J_{j_2} are both late in S' , i.e., $C_{j_1}(S') > d_{j_1}$ and $C_{j_2}(S') > d_{j_2}$. Then it follows that

$$\begin{aligned} T_{j_1}(S') + T_{j_2}(S') &= C_{j_1}(S') - d_{j_1} + C_{j_2}(S') - d_{j_2} \\ &= C_{j_2}(S^*) - d_{j_1} + C_{j_1}(S^*) - d_{j_2} \\ &= C_{j_1}(S^*) - d_{j_1} + C_{j_2}(S^*) - d_{j_2} \\ &\leq T_{j_1}(S^*) + T_{j_2}(S^*). \end{aligned}$$

Secondly, suppose J_{j_1} is on time in S' , i.e. $C_{j_1}(S') \leq d_{j_1}$. Using $d_{j_1} \leq d_{j_2}$, we obtain

$$\begin{aligned} T_{j_1}(S') + T_{j_2}(S') &= 0 + \max\{C_{j_2}(S') - d_{j_2}, 0\} \\ &= \max\{C_{j_1}(S^*) - d_{j_2}, 0\} \\ &\leq \max\{C_{j_1}(S^*) - d_{j_1}, 0\} \\ &= T_{j_1}(S^*) \\ &\leq T_{j_1}(S^*) + T_{j_2}(S^*). \end{aligned}$$

Thirdly, suppose J_{j_2} is on time in S' , i.e. $C_{j_2}(S') \leq d_{j_2}$. Using $C_{j_1}(S^*) > C_{j_2}(S^*)$, we obtain

$$\begin{aligned} T_{j_1}(S') + T_{j_2}(S') &= \max\{C_{j_1}(S') - d_{j_1}, 0\} + 0 \\ &= \max\{C_{j_2}(S^*) - d_{j_1}, 0\} \\ &\leq \max\{C_{j_1}(S^*) - d_{j_1}, 0\} \\ &= T_{j_1}(S^*) \\ &\leq T_{j_1}(S^*) + T_{j_2}(S^*). \end{aligned}$$

Hence, performing the described swap does neither increase the maximum lateness nor the total tardiness. Therefore, as in (i), we can perform such exchanges sequentially in order to obtain an optimal permutation schedule with jobs scheduled in the given earliest due date order. Hence, any given earliest due date ordering of the jobs is optimal. \square

Appendix B: Proof of Theorem 19

Suppose that jobs are indexed in an earliest due date order, i.e. $d_1 \leq d_2 \leq \dots \leq d_n$. As in Sect. 3, the modified dynamic program tries to schedule these jobs one by one, but it is also allowed to leave jobs out that will be late anyway, which then will be scheduled in the end after all on-time jobs. This procedure is justified by the following lemma.

Lemma 20 *Consider a PFB instance I without release dates to minimize the weighted number of late jobs. Suppose jobs are indexed by an earliest due date ordering. Then there exists an optimal permutation schedule in which the on-time jobs are scheduled in the order of their indices.*

Proof By Theorem 16, there exists an optimal permutation schedule S^* . Let \mathcal{J} be the set of on-time jobs in S^* . Let $I_{\mathcal{J}}$ be the modified PFB instance that contains only the jobs in \mathcal{J} . By ignoring all jobs outside of \mathcal{J} , schedule S^* defines a feasible permutation schedule $S^*_{\mathcal{J}}$ for $I_{\mathcal{J}}$ that has maximum lateness 0. By Theorem 17 (ii), there exists also a permutation schedule $S'_{\mathcal{J}}$ for $I_{\mathcal{J}}$ in which jobs are scheduled in the order of their indices and that has maximum lateness 0. By scheduling the remaining jobs arbitrarily after the on-time jobs, $S'_{\mathcal{J}}$ can be extended to a schedule S' for I which has the same weighted number of late jobs as S^* and in which all on-time jobs are scheduled in the order of their indices. \square

Recall that for a PFB instance I and a job index $j \in [n]$, I_j denotes the modified instance containing only jobs J_1, \dots, J_j . Also, recall the notation $\Gamma = \Gamma_1 \times \Gamma_2 \times \dots \times \Gamma_m$ and $\mathcal{B} = [b_1] \times [b_2] \times \dots \times [b_m]$, where \times is the standard Cartesian product. The variables of the new dynamic program are defined almost as in Sect. 3. However, in contrast, we say that a permutation schedule S for instance I_j corresponds to vectors

$$t = (t_1, t_2, \dots, t_i, \dots, t_{m-1}, t_m) \in \Gamma$$

and

$$k = (k_1, k_2, \dots, k_i, \dots, k_{m-1}, k_m) \in \mathcal{B}$$

if the last on-time job in S finishes machine M_i at time t_i and is processed in a batch with exactly k_i on-time jobs there for all $i \in [m]$. If there are no on-time jobs, then S cannot correspond to any vectors t and k . Note that we do not require that J_j itself is on time, just that there is some on-time job. The difference to Sect. 3 is that we ignore all late jobs in this definition. Therefore, only for the on-time jobs it is assumed that they are processed in the order of job indices.

Similar to Sect. 3, we define $\tilde{S}(j, t, k)$ to be the set of feasible permutation schedules S for instance I_j satisfying the following properties:

1. the on-time jobs in S are ordered by their indices,
2. S corresponds to t and k , and
3. S is Γ -active.

The variables of our new dynamic program are defined as follows. For $j \in [n]$, $t \in \Gamma$, and $k \in \mathcal{B}$, let

$$w(j, t, k) = w(j, t_1, t_2, \dots, t_m, k_1, k_2, \dots, k_m)$$

be the minimum weighted number of late jobs of a schedule $S \in \tilde{S}(j, t, k)$. If no such schedule exists, the value of w is defined to be $+\infty$.

Using the above definitions as well as Lemmas 9 and 20, if there is at least one on-time job in the optimal schedule, the minimum weighted number of late jobs is given by $\min_{t,k} w(n, t, k)$, $t \in \Gamma$, $k \in \mathcal{B}$. In Lemmas 21 and 22 (below) we provide formulas to recursively compute the values $w(j, t, k)$, $j = 1, 2, \dots, n$, $t \in \Gamma$, $k \in \mathcal{B}$. In total, we then obtain Algorithm 2 as our modified dynamic program.

Algorithm 2

Input: A PFB instance without release dates with jobs indexed in an earliest due date ordering.

Output: A schedule minimizing the weighted number of late jobs.

- 1: For each $t \in \Gamma$ and $k \in \mathcal{B}$, compute $w(1, t, k)$ according to Lemma 21.
 - 2: For each $j = 2, 3, \dots, n$, $t \in \Gamma$, and $k \in \mathcal{B}$, compute $w(j, t, k)$ according to Lemma 22. If $w(j, t, k) < +\infty$, also store which job was the last on-time job added to the schedule and its corresponding t and k values.
 - 3: If there are t and k with $w(n, t, k) < +\infty$, find the values of t and k that minimize $w(n, t, k)$ and start a backtracking procedure from them to reconstruct the optimal schedule using the information stored in Step 2. Otherwise, return an arbitrary feasible schedule.
-

We now turn to providing and proving the correctness of the recursive formula to compute the values $w(j, t, k)$, $j = 1, 2, \dots, n$, $t \in \Gamma$, $k \in \mathcal{B}$. As the recursion is slightly more complicated than in Sect. 3, we first introduce two kinds of auxiliary variables x and y for our dynamic program. These are then used to formulate and prove Lemmas 21 and 22, which deal with the starting values $w(1, t, k)$ and the actual recurrence relation, respectively, in analogy to Lemmas 10 and 11. Finally, by connecting these results, we obtain the correctness and runtime of Algorithm 2 as stated in Theorem 19.

For any $j \in [n]$, $t_i \in \Gamma_i$, and $k_i \in [b_i]$, $i \in [m]$, let

$$x(j, t, k) = \begin{cases} \sum_{j'=1}^{j-1} w_{j'}, & \text{if (i)–(iv) hold,} \\ +\infty, & \text{otherwise,} \end{cases} \tag{5}$$

where

- (i) $k_i = 1$ for all $i \in [m]$,
- (ii) $t_1 \geq p_1$,
- (iii) $t_{i+1} \geq t_i + p_{i+1}$ for all $i \in [m - 1]$, and
- (iv) $t_m \leq d_j$.

The values of x can be interpreted as the objective value of a schedule in $\tilde{S}(j, t, k)$ for I_j that only schedules J_j on time and all other jobs late.

For $j > 1$, $t_i \in \Gamma_i$, and $k_i \in [b_i]$, $i \in [m]$, let

$$y(j, t, k) = \begin{cases} \min\{w(j - 1, t', k') \mid (**)\}, & \text{if (ii)–(iv) hold,} \\ +\infty, & \text{otherwise,} \end{cases} \quad (6)$$

where the minimum over the empty set is defined to be $+\infty$. Here, $(**)$ is given by conditions

- (v) $t'_i \in \Gamma_i$,
- (vi) $k'_i \in [b_i]$,
- (vii) if $k_i = 1$, then $t'_i \leq t_i - p_i$, and
- (viii) if $k_i > 1$, then $t'_i = t_i$ and $k'_i = k_i - 1$,

for all $i \in [m]$. The values of y can be interpreted as the best possible objective value of a schedule in $\tilde{S}(j, t, k)$ that schedules J_j and at least one other job on time.

Analogous to Lemma 10, we obtain the following lemma.

Lemma 21 For $t_i \in \Gamma_i$ and $k_i \in [b_i]$, $i \in [m]$, we have $w(1, t, k) = x(1, t, k)$.

Proof As in the proof of Lemma 10, conditions (i)–(iii) are necessary for the existence of a schedule in $\tilde{S}(1, t, k)$. Also, since a schedule for I_1 can only correspond to t and k if J_1 is on time, (iv) is necessary as well. Conversely, if (i)–(iv) are satisfied, then the vector t of completion times uniquely defines a feasible schedule $S \in \tilde{S}(1, t, k)$ by $c_{i1}(S) = t_i$. Note that S is the only schedule in $\tilde{S}(1, t, k)$ and has objective value 0 and therefore, clearly, is optimal. \square

Analogous to Lemma 11, we now turn to the recurrence formula to calculate w for $j > 1$ from the values of w for $j - 1$.

Lemma 22 For $j \geq 2$, $t_i \in \Gamma_i$, and $k_i \in [b_i]$, $i \in [m]$, the values of w are given by

$$w(j, t, k) = \min\{x(j, t, k), y(j, t, k), w(j - 1, t, k) + w_j\}.$$

Roughly speaking, for scheduling J_j there are three possibilities: either job J_j is scheduled as the only on-time job, as one of several on-time jobs, or as a late job. In the first two cases, $w(j, t, k)$ equals $x(j, t, k)$ or $y(j, t, k)$, respectively,

while in the third case it equals $w(j - 1, t, k) + w_j$, as we will see in the proof.

Proof of Lemma 22: We first prove the “ \geq ” direction. If the left-hand side equals $+\infty$, then this direction follows immediately. Otherwise, by definition of w , there exists a schedule S for I_j in $\tilde{S}(j, t, k)$ with objective value $w(j, t, k)$. Schedule S naturally defines a feasible permutation schedule S' for instance I_{j-1} by ignoring job J_j . We distinguish three cases.

Firstly, if J_j is not on time in S , then S' is in $\tilde{S}(j - 1, t, k)$ and has an objective value of $w(j, t, k) - w_j$ because J_j is not contained in instance J_{j-1} . Therefore, by definition of $w(j - 1, t, k)$, it follows that $w(j - 1, t, k) \leq w(j, t, k) - w_j$, which implies the “ \geq ” direction in the first case.

Secondly, if J_j is the only on-time job in S , then (i) must hold because S corresponds to k and only on-time jobs are counted in this definition. Furthermore, (ii) and (iii) must hold because S is feasible and corresponds to t . Finally (iv) holds because J_j is on time. Hence, we obtain $x(j, t, k) = \sum_{j'=1}^{j-1} w_{j'} = w(j, t, k)$ in this case, which implies the “ \geq ” direction in the second case.

Thirdly, if J_j is one of at least two on-time jobs in S , then S' corresponds to unique vectors t' and k' . Hence, we obtain $S' \in \tilde{S}(j - 1, t', k')$ and, therefore,

$$\begin{aligned} w(j, t, k) &= \sum_{j'=1}^j w_{j'} U_{j'}(S) \\ &= \sum_{j'=1}^{j-1} w_{j'} U_{j'}(S') \\ &\geq w(j - 1, t', k'). \end{aligned}$$

Analogous arguments to the proof of Lemma 11 show that t' and k' satisfy $(**)$. Moreover, as in the second case, (ii)–(iv) hold because S is feasible, corresponds to t , and J_j is on time. Therefore, we have

$$w(j, t, k) \geq w(j - 1, t', k') \geq y(j, t, k),$$

which completes the “ \geq ” direction.

Now we prove the “ \leq ” direction. If the right-hand side equals $+\infty$, then this direction follows immediately. Otherwise, we again distinguish three cases.

Firstly, suppose the minimum on the right-hand side is attained by the term $w(j - 1, t, k) + w_j$. Since this must be finite, there is a schedule $S' \in \tilde{S}(j - 1, t, k)$ for instance I_{j-1} with objective value $w(j - 1, t, k)$. By scheduling J_j as a late job, S' can be extended to a schedule S that has objective value $w(j - 1, t, k) + w_j$. Moreover, since J_j is late, S still corresponds to t and k , i.e. $S \in \tilde{S}(j, t, k)$, which implies $w(j, t, k) \leq w(j - 1, t, k) + w_j$. Hence, the “ \leq ” direction follows in this case.

Secondly, suppose the minimum on the right-hand side is attained by the term $x(j, t, k)$. Since this must be finite, we obtain that (i)–(iv) hold. Hence, the schedule S which schedules J_j such that $c_{ij} \approx t_i$ and all other jobs late is feasible for I_j , an element of $\tilde{S}(j, t, k)$, and has objective value $x(j, t, k) = \sum_{j'=1}^{j-1} w_{j'}$. This implies $w(j, t, k) \leq x(j, t, k)$. Hence, the “ \leq ” direction follows in this case.

Thirdly, suppose the minimum on the right-hand side is attained by the term $y(j, t, k)$. Since this must be finite, we obtain that (ii)–(iv) hold. Let t' and k' be the minimizers in the definition of $y(j, t, k)$. Again due to finiteness, there must be a schedule $S' \in \tilde{S}(j-1, t', k')$ for I_{j-1} with objective value $w(j-1, t', k') = y(j, t, k)$. By ignoring late jobs in S' (which can be scheduled arbitrary late), we can use the same arguments as in the proof of Lemma 11 to extend S' to schedule $S \in \tilde{S}(j, t, k)$ for I_j . Due to (iv), J_j is on time in S . Hence, S has the same objective value as S' , namely $y(j, t, k)$. Thus, it follows that $w(j, t, k) \leq y(j, t, k)$, which completes the “ \leq ” direction. \square

Analogous to Theorem 12, we now deduce the desired theorem by proving correctness and running time of Algorithm 2.

Theorem 19 *For a PFB without release dates and a fixed number m of machines, the weighted number of late jobs can be minimized in $\mathcal{O}(n^{m^2+2m-1})$ time.*

Proof Without loss of generality, jobs are indexed in an earliest due date ordering. By Lemmas 21 and 22, Algorithm 2 correctly computes the values of function w . If there is a feasible schedule where at least one job is on time, we obtain due to Lemmas 9 and 20 that the minimum weighted number of late jobs is exactly $\min_{t,k} w(n, t, k)$, where the minimum is taken over each $t \in \Gamma$ and $k \in \mathcal{B}$. In this case the backtracking procedure in Step 3 retrieves an optimal schedule. If, however, in every feasible schedule all jobs are late, then all values of w will be $+\infty$. In this case, any feasible schedule is optimal, with objective value $\sum_{j=1}^n w_n$. Hence, in any case, the schedule returned by Algorithm 2 is optimal.

Recall from the proof of Theorem 12 that $|\Gamma| \cdot |\mathcal{B}| \leq n^{\frac{m^2}{2} + \frac{5m}{2}}$. Therefore, Step 1 of Algorithm 2 involves at most $n^{\frac{m^2}{2} + \frac{5m}{2}}$ iterations, each of which can be performed in constant time because m is fixed.

Similarly, Step 2 involves at most $(n-1)n^{\frac{m^2}{2} + \frac{5m}{2}} \leq n^{\frac{m^2}{2} + \frac{5m}{2} + 1}$ iterations. In each of these iterations, computing $x(j, t, k)$ with (5) can be done in constant time since m is fixed. The same holds for computing $w(j, t, k)$ using Lemma 22, once $y(j, t, k)$ is known. However, the computation of $y(j, t, k)$ by applying (6) is the bottleneck of this step. In full analogy to the proof of Theorem 12, we obtain that the total time complexity of computing all y -values is bounded by $\mathcal{O}(n \prod_{i=1}^m b_i |\Gamma_i|^2)$. Due to vanishing release

dates, we have $|\Gamma_i| \leq n^i$. Comparing with the estimation (4) in the proof of Theorem 12, this results in a runtime bound of $\mathcal{O}(n^{m^2+2m+1})$.

However, the following argument allows us to reduce this runtime by another factor of n^2 , resulting in $\mathcal{O}(n^{m^2+2m-1})$. Even though we do not know the optimal job permutation beforehand, we know by Sung et al. (2000, Theorem 1) that it is optimal to use the so-called last-only-empty batching on the first machine M_1 , that is, to completely fill all batches on M_1 , except for the last batch, which might contain less jobs if n is not a multiple of b_1 . This has two consequences. First, it suffices to consider the $\lceil \frac{n}{b_1} \rceil$ possible completion times $\Gamma'_1 = \{p_1, 2p_1, \dots, \lceil \frac{n}{b_1} \rceil p_1\}$ on M_1 , instead of all elements of Γ_1 . Second, for given $t_1 \in \Gamma'_1$ and $k_1 \in [b_1]$, the possible previous values t'_1 and k'_1 are uniquely determined. Thus, in our runtime estimate, we may replace $|\Gamma_1|$ with $|\Gamma'_1|$ and leave out the square, such that instead of the term $b_1 |\Gamma_1|^2$ we now have $b_1 |\Gamma'_1| \in \mathcal{O}(b_1 \frac{n}{b_1}) = \mathcal{O}(n)$. Previously, this term was bounded by $\mathcal{O}(n^3)$, resulting in the claimed improvement.

Finally, in Step 3, we take the minimum over at most $n^{\frac{m^2}{2} + \frac{5m}{2}}$ many values. The backtracking and reconstruction of the final schedule can be done in time $\mathcal{O}(n)$ for constant m .

Hence, in total, Step 2 is the bottleneck and the total time complexity is $\mathcal{O}(n^{m^2+2m-1})$. \square

References

Ackermann, H., Heydrich, S., & Weiß, C. (2020+). Analyzing and optimizing the throughput of a pharmaceutical production process. In *Operations research proceedings 2019*, Springer, accepted.

Ahmadi, J. H., Ahmadi, R. H., Dasu, S., & Tang, C. S. (1992). Batching and scheduling jobs on batch and discrete processors. *Operations Research*, 40(4), 750–763.

Baptiste, P. (2000). Batching identical jobs. *Mathematical Methods of Operations Research*, 52(3), 355–367.

Brucker, P. (2007). *Scheduling Algorithms* (5th ed.). Berlin: Springer.

Brucker, P., & Knust, S. (2009). Complexity results for scheduling problems. <http://www.informatik.uni-osnabrueck.de/knust/class/>. Accessed on September 11, 2018.

Brucker, P., & Shakhlevich, N. V. (2011). A polynomial-time algorithm for a flow-shop batching problem with equal-length operations. *Journal of Scheduling*, 14(4), 371–389.

Brucker, P., Gladky, A., Hoogeveen, H., Kovalyov, M. Y., Potts, C. N., Tautenhahn, T., et al. (1998). Scheduling a batching machine. *Journal of Scheduling*, 1(1), 31–54.

Chen, Z., Zheng, X., Zhou, S., Liu, C., & Chen, H. (2019). Quantum-inspired ant colony optimisation algorithm for a two-stage permutation flow shop with batch processing machines. *International Journal of Production Research*.

Condotta, A., Knust, S., & Shakhlevich, N. V. (2010). Parallel batch scheduling of equal-length jobs with release and due dates. *Journal of Scheduling*, 13(5), 463–477.

Diessel, E., & Ackermann, H. (2019). Domino sequencing: Scheduling with state-based sequence-dependent setup times. *Operations Research Letters*, 47(4), 274–280.

- Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2), 117–129.
- Graham, R. L., Lawler, E. L., Lenstra, J. K., & Rinnooy Kan, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. In P. L. Hammer, E. L. Johnson, & B. H. Korte (Eds.), *Discrete optimization II, annals of discrete mathematics* (Vol. 5, pp. 287–326). Amsterdam: Elsevier.
- Hertrich, C. (2018). Scheduling a proportionate flow shop of batching machines. Master thesis, Technische Universität Kaiserslautern. <http://nbn-resolving.de/urn:nbn:de:hbz:386-kluedo-54968>.
- Hochbaum, D. S., & Shamir, R. (1990). Minimizing the number of tardy job units under release time constraints. *Discrete Applied Mathematics*, 28(1), 45–57.
- Hochbaum, D. S., & Shamir, R. (1991). Strongly polynomial algorithms for the high multiplicity scheduling problem. *Operations Research*, 39(4), 648–653.
- Ikura, Y., & Gimple, M. (1986). Efficient scheduling algorithms for a single batch processing machine. *Operations Research Letters*, 5(2), 61–65.
- Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1), 61–68.
- Lee, C. Y., Uzsoy, R., & Martin-Vega, L. A. (1992). Efficient algorithms for scheduling semiconductor burn-in operations. *Operations Research*, 40(4), 764–775.
- Li, D., Meng, X., Liang, Q., & Zhao, J. (2015). A heuristic-search genetic algorithm for multi-stage hybrid flow shop scheduling with single processing machines and batch processing machines. *Journal of Intelligent Manufacturing*, 26(5), 873–890.
- Li, F., Zhang, Y., Wei, H., & Lai, X. (2019). Integrated problem of soaking pit heating and hot rolling scheduling in steel plants. *Computers & Operations Research*, 108, 238–246.
- Mnich, M., & van Bevern, R. (2018). Parameterized complexity of machine scheduling: 15 open problems. *Computers & Operations Research*, 100, 254–261.
- Mnich, M., & Wiese, A. (2015). Scheduling and fixed-parameter tractability. *Mathematical Programming*, 154(1), 533–562.
- Ng, C. T., & Kovalyov, M. Y. (2007). Batching and scheduling in a multi-machine flow shop. *Journal of Scheduling*, 10(6), 353–364.
- Panwalkar, S. S., Smith, M. L., & Koulamas, C. (2013). Review of the ordered and proportionate flow shop scheduling research. *Naval Research Logistics (NRL)*, 60(1), 46–55.
- Passchyn, W., & Spieksma, F. C. R. (2019). Scheduling parallel batching machines in a sequence. *Journal of Scheduling*, 22(3), 335–357.
- Potts, C. N., Strusevich, V. A., & Tautenhahn, T. (2001). Scheduling batches with simultaneous job processing for two-machine shop problems. *Journal of Scheduling*, 4(1), 25–51.
- Rachamadugu, R. V., Vepsalainen, A., & Morton, T. E. (1982). Scheduling in proportionate flowshops. Tech. Rep. CMU-RI-TR-83-10, Carnegie Mellon University, Pittsburgh, PA.
- Sung, C. S., & Kim, Y. H. (2003). Minimizing due date related performance measures on two batch processing machines. *European Journal of Operational Research*, 147(3), 644–656.
- Sung, C. S., & Yoon, S. H. (1997). Minimizing maximum completion time in a two-batch-processing-machine flowshop with dynamic arrivals allowed. *Engineering Optimization*, 28(3), 231–243.
- Sung, C. S., Kim, Y. H., & Yoon, S. H. (2000). A problem reduction and decomposition approach for scheduling for a flowshop of batch processing machines. *European Journal of Operational Research*, 121(1), 179–192.
- Tan, Y., Mönch, L., & Fowler, J. W. (2018). A hybrid scheduling approach for a two-stage flexible flow shop with batch processing machines. *Journal of Scheduling*, 21(2), 209–226.
- Weiß, C., Knust, S., Shakhlevich, N., & Waldherr, S. (2016). The assignment problem with nearly monge arrays and incompatible partner indices. *Discrete Applied Mathematics*, 211, 183–203.
- Zhou, S., Li, X., Chen, H., & Guo, C. (2016). Minimizing makespan in a no-wait flowshop with two batch processing machines using estimation of distribution algorithm. *International Journal of Production Research*, 54(16), 4919–4937.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.